## EXPERIMENT NO- 4

**AIM:** Implementation and analysis of Merge Sort

## PROBLEM STATEMENT :

**a.** WAP to sort given numbers using Merge Sort algorithm.

**Resource Required**: Pentium IV, Turbo C, Printer, Printout Stationary

## THEORY:

1. **Merge sort: -**

Merge sort is based on the divide-and-conquer paradigm. Its worst-case running time has a lower order of growth than insertion sort. Since we are dealing with sub problems, we state each sub problem as sorting a subarray *A*[*p* .. *r*]. Initially, *p* = 1 and *r* = *n*, but these values change as we recurse through sub problems.

To sort *A* [*p, r*]:

## Divide Step

If a given array *A* has zero or one element, simply return; it is already sorted. Otherwise, split *A*[*p* .. *r*] into two subarrays*A*[*p* .. *q*] and *A*[*q* + 1 .. *r*], each containing about half of the elements of *A*[*p* .. *r*]. That is, *q* is the halfway point of *A*[*p* .. *r*].

## Conquer Step

Conquer by recursively sorting the two subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*].

## Combine Step

Combine the elements back in *A*[*p* .. *r*] by merging the two sorted subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (*A*, *p*, *q*, *r*).

Note that the recursion bottoms out when the subarray has just one element, so that it is trivially sorted.

## Algorithm: Merge Sort

void mergesort(int arr[],int low,int high)

{

if(low<high)

{

int mid = (low+high)/2;

mergesort(arr,low,mid);

mergesort(arr,mid+1,high);

merge(arr,low,mid,high);

}

}

void merge(int arr[],int low,int mid,int high)

{

int n1 = mid-low+1, n2 =high-mid;

int L1[n1],L2[n2];

for(int i=0;i<n1;i++)

L1[i] = arr[i+low];

for(int i=0;i<n2;i++)

L2[i] = arr[i+mid+1];

int i=0,j=0;

for(int k=low;k<=high;k++)

{

if(L1[i]>L2[j])

{

arr[k] = L2[j++];

}

else{

arr[k] = L1[i++];

}

}

}

**CONCLUSION:** The total running time of Merge sort algorithm is O (*n* lg *n*), which is asymptotically optimal like Heap sort, Merge sort has a guaranteed *n* lg *n* running time. Merge sort required (*n*) extra space. Merge is not in-place algorithm.

**Code**:

#include<stdio.h>

void merge(int arr[],int low,int mid,int high);

void mergesort(int arr[],int low,int high);

int main(){

int n;

printf("Enter number of elements you have in your array\n");

scanf("%d",&n);

int arr[n];

printf("Enter Elements of your array \n");

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

}

mergesort(arr,0,n-1);

printf("Sorted array: \n");

for(int i=0;i<n;i++){

printf("%d \t",arr[i]);

}

return 0;

}

void mergesort(int arr[],int low,int high)

{

if(low<high)

{

int mid = (low+high)/2;

mergesort(arr,low,mid);

mergesort(arr,mid+1,high);

merge(arr,low,mid,high);

}

}

void merge(int arr[],int low,int mid,int high)

{

int n1 = mid-low+1, n2 =high-mid;

int L1[n1],L2[n2];

for(int i=0;i<n1;i++)

L1[i] = arr[i+low];

for(int i=0;i<n2;i++)

L2[i] = arr[i+mid+1];

int i=0,j=0;

for(int k=low;k<=high;k++)

{

if(L1[i]>L2[j])

{

arr[k] = L2[j++];

}

else{

arr[k] = L1[i++];

}

}

}

**Output**:

